# <u>Big Data Project</u>

**<u>Aharon Rabson</u>**

## Datasource

<u>Dataset used</u>: [Student Mental Health Analysis]

<u>Source</u>: Kaggle.com

I chose this dataset because mental health analysis, especially in an online learning context, is highly relevant and provides meaningful insights for educational institutions. Furthermore, I wanted more experience in the medical niche as that is where my future interests as a data scientist lie.

## Data Pipeline Overview

My pipeline begins with NiFi, which ingests the student mental health dataset from a GitHub repository directly into HDFS. Once stored in HDFS, Hive utilizes this data by creating an external table, enabling efficient querying. PySpark then loads the data from Hive, processes it, performs exploratory analysis, and trains a Logistic Regression model. Finally, the accuracy metrics of this model are stored in HBase for persistent, fast retrieval.

## Issues Encountered:

- **Issue**: PySpark read the header row as data due to Hive's external table limitation.

  **Solution**: Manually filtered the header row in PySpark after loading the table.

- **Issue**: NiFi initially failed to write files into HDFS due to permission errors.

  **Solution**: Updated NiFi's Hadoop configuration XML paths and ensured proper permissions.

- **Issue**: Difficulty starting HBase Thrift server (zookeeper nodes not found).

  **Solution**: Restarted the HBase Master and RegionServer processes explicitly.

# Data Ingestion (NiFi)

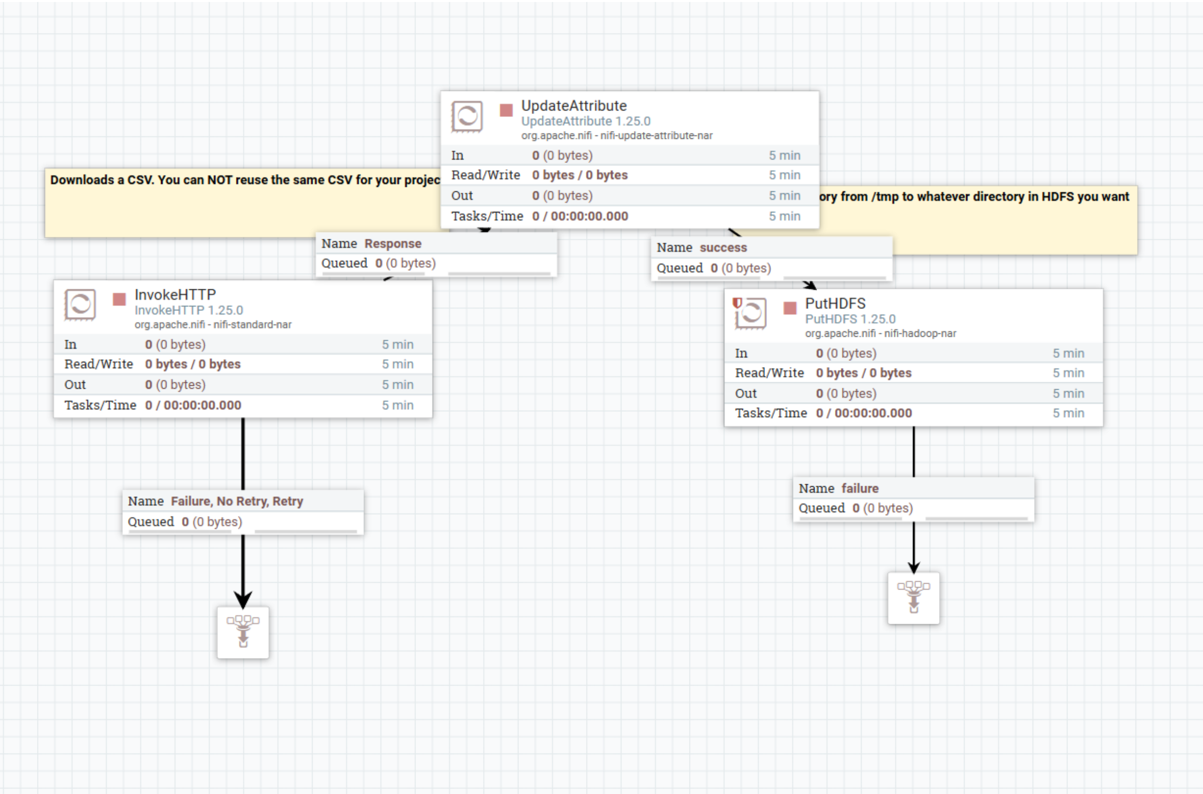



*Figure 1: NiFi workflow*

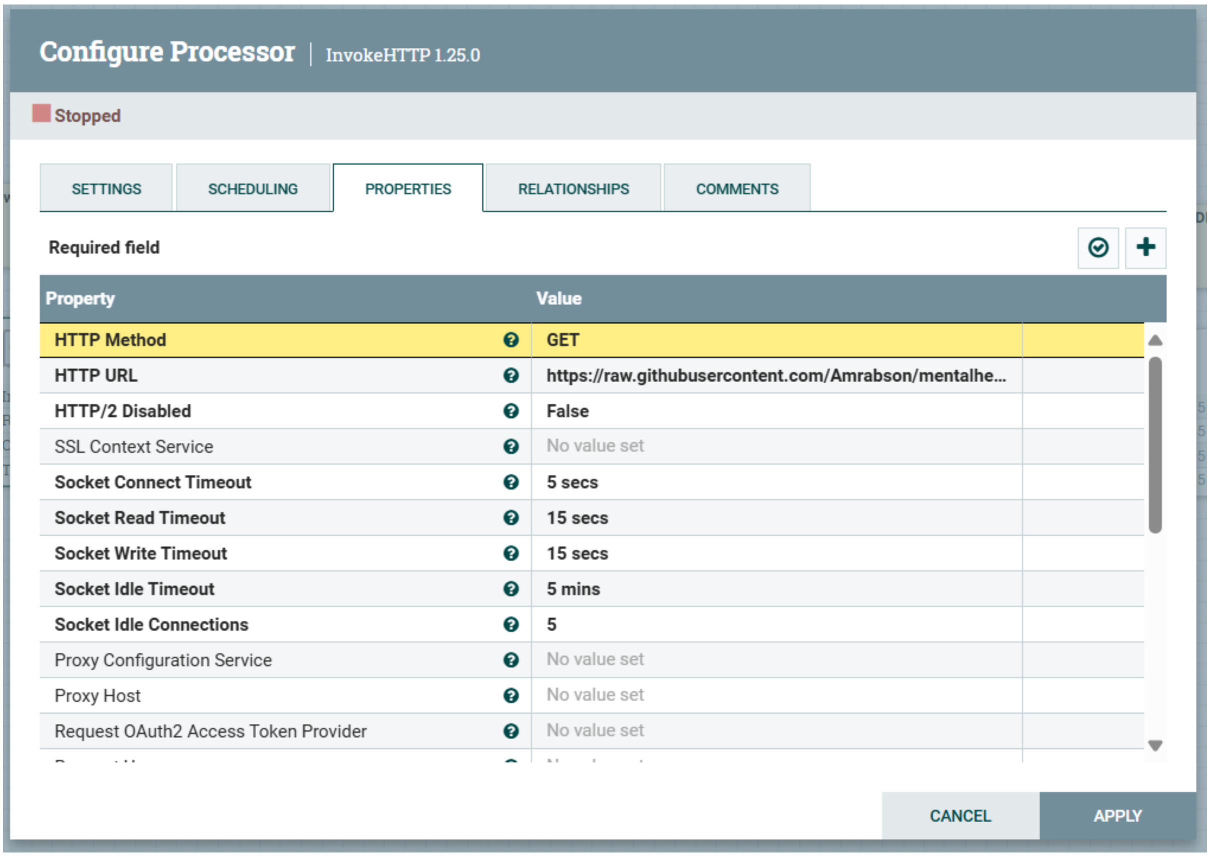



*Figure 2: Configuration of InvokeHTTP attribute*

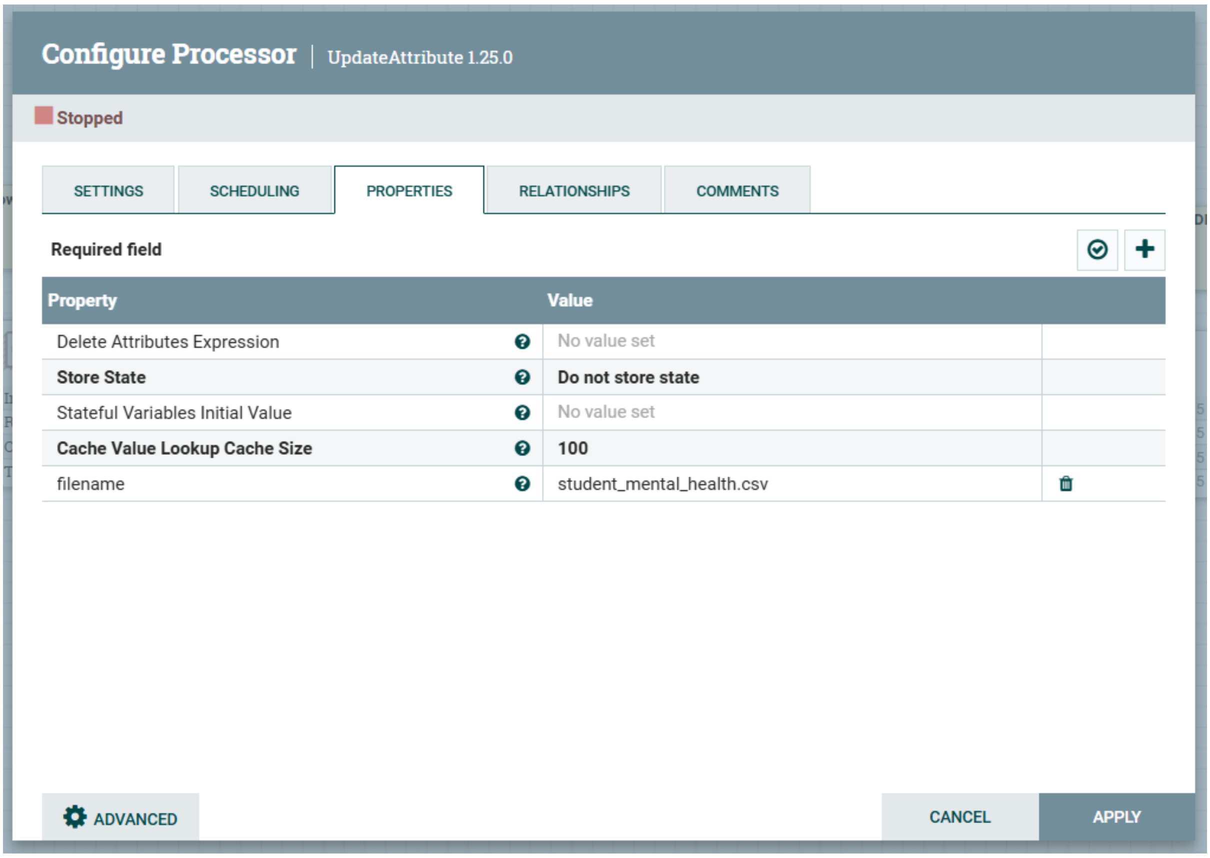



*Figure 3: Configuration of UpdateAttribute processor to rename file*

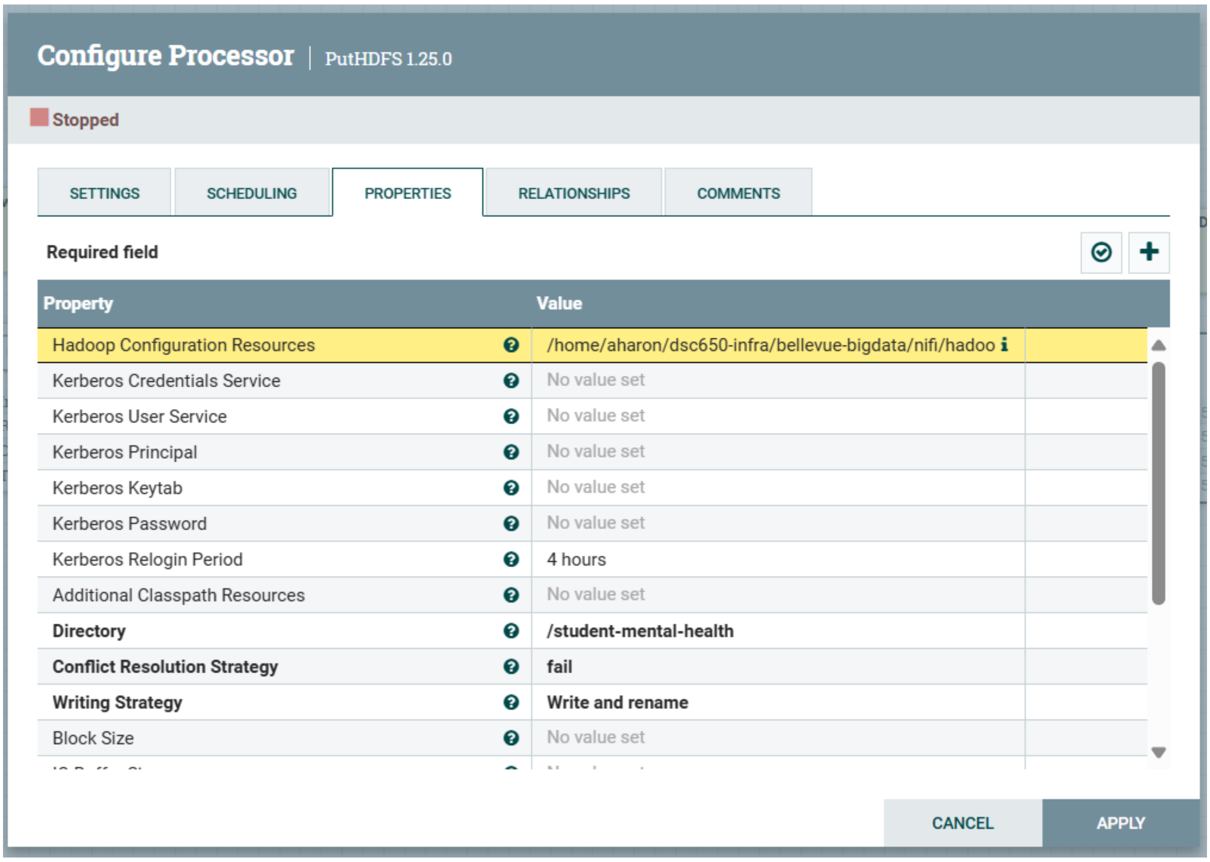



*Figure 4: Configuration of PutHDFS attribute*

HDFS command to confirm data:

```
hdfs dfs -ls /student-mental-health
```

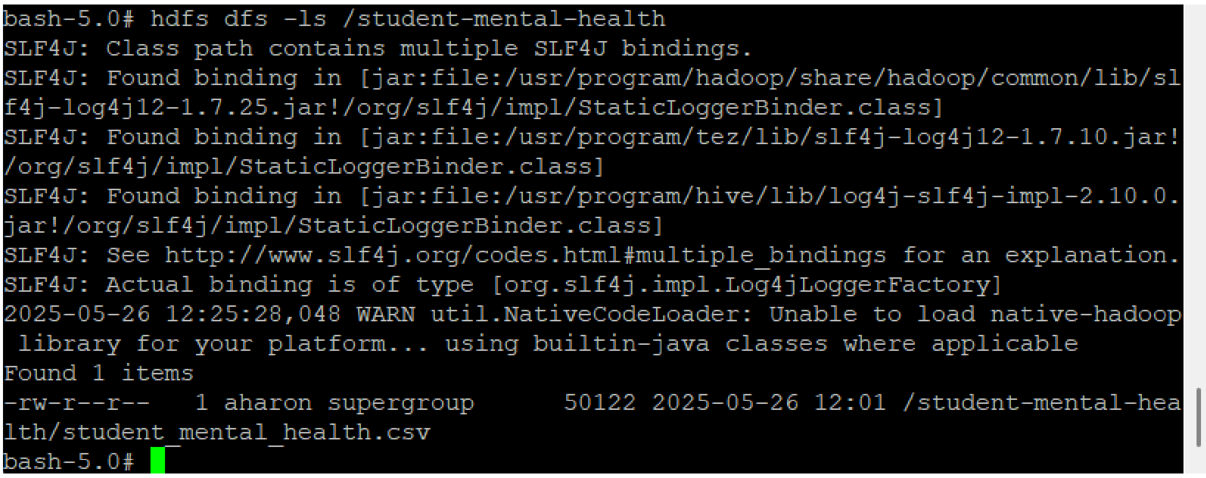



*Figure 5: Confirmation of successful transfer of dataset into HDFS*

# HDFS to Hive

Open the Hive CLI:

```
hive
```

```
bash-5.0# hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/program/hive/lib/log4j-slf4j-impl-2.10.0.
jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/hadoop/share/hadoop/common/lib/sl
f4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/tez/lib/slf4j-log4j12-1.7.10.jar!
/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive Session ID = 7434b7b6-cfdf-4e9c-82cb-d35afe88fc6f

Logging initialized using configuration in file:/usr/program/hive/conf/hive-log4
j2.properties Async: true
Hive Session ID = b0f26e15-0746-4772-a64d-1651b920b236
2025-05-26 15:44:14,086 INFO  [Tez session start thread] client.RMProxy: Connect
ing to ResourceManager at master/172.28.1.1:8032
hive> 2025-05-26 15:44:15,094 INFO  [pool-7-thread-1] client.RMProxy: Connecting
 to ResourceManager at master/172.28.1.1:8032
```

*Figure 6: Initializing Hive*

Creating an External Table in Hive:

```
CREATE DATABASE IF NOT EXISTS mental_health;
USE mental_health;

CREATE EXTERNAL TABLE student_mental_health(
    Name STRING,
    Gender STRING,
    Age INT,
    Education_Level STRING,
    Screen_Time STRING,
    Sleep_Duration STRING,
    Physical_Activity STRING,
    Stress_Level STRING,
    Anxious_Before_Exams STRING,
    Academic_Performance_Change STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/student-mental-health'
tblproperties ("skip.header.line.count"="1");
```

```
    > CREATE DATABASE IF NOT EXISTS mental_health;
OK
Time taken: 1.549 seconds
hive> USE mental_health;
OK
Time taken: 0.045 seconds
hive>
    > CREATE EXTERNAL TABLE student_mental_health(
    >     Name STRING,
    >     Gender STRING,
    >     Age INT,
    >     Education_Level STRING,
    >     Screen_Time STRING,
    >     Sleep_Duration STRING,
    >     Physical_Activity STRING,
    >     Stress_Level STRING,
    >     Anxious_Before_Exams STRING,
    >     Academic_Performance_Change STRING
    > )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE
    > LOCATION '/student-mental-health'
    > tblproperties ("skip.header.line.count"="1");
OK
Time taken: 0.618 seconds
hive>
```

*Figure 7: Creating external table in Hive*

Checks data loaded in Hive:

```
SELECT * FROM student_mental_health LIMIT 10;
```

```
hive> SELECT * FROM student_mental_health LIMIT 10;
OK
Aarav   Male    15      Class 8 7.1     8.9     9.3     Medium  No      Same
Meera   Female  25      MSc     3.3     5.0     0.2     Medium  No      Same
Ishaan  Male    20      BTech   9.5     5.4     6.2     Medium  No      Same
Aditya  Male    20      BA      10.8    5.6     5.5     High    Yes     Same
Anika   Female  17      Class 11        2.8     5.4     3.1     Medium  Yes     S
ame
Aditya  Male    23      MSc     8.6     8.4     0.1     Low     No      Improved
Vivaan  Male    22      MTech   3.6     6.6     0.5     Medium  Yes     Improved
Arjun   Male    25      MTech   7.0     4.7     4.5     Medium  No      Same
Sai     Male    20      BA      4.8     5.0     7.9     Medium  No      Improved
Aadhya  Female  16      Class 9 8.9     8.4     7.8     Low     Yes     Improved
Time taken: 3.623 seconds, Fetched: 10 row(s)
hive>
```

*Figure 8: Sample output showing successful table creation*

# Hive to Pyspark

Launching PySpark with Hive support:

```
pyspark --conf spark.sql.catalogImplementation=hive
```

```
    > bash-5.0# pyspark --conf spark.sql.catalogImplementation=hive
Python 3.7.10 (default, Mar  2 2021, 09:06:08)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/program/spark/jars/slf4j-log4j12-1.7.30.j
ar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/hadoop/share/hadoop/common/lib/sl
f4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
0    [main] WARN  org.apache.hadoop.util.NativeCodeLoader  - Unable to load nati
ve-hadoop library for your platform... using builtin-java classes where applicab
le
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
1157 [Thread-4] WARN  org.apache.hadoop.hive.conf.HiveConf  - HiveConf of name h
ive.strict.managed.tables does not exist
1157 [Thread-4] WARN  org.apache.hadoop.hive.conf.HiveConf  - HiveConf of name h
ive.create.as.insert.only does not exist
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.0.0
      /_/

Using Python version 3.7.10 (default, Mar  2 2021 09:06:08)
SparkSession available as 'spark'.
>>>
```

*Figure 9: Initializing PySpark with Hive support*

Query the Hive table using PySpark:

```
spark.sql("USE mental_health")

df = spark.sql("SELECT * FROM student_mental_health")


# Drop header row: 'name' column will literally be
'Name' in header

df_clean = df.filter("name != 'Name'")

df_clean.show()
```

**Note:** Spark doesn't honor skip.header.line.count. from Hive. This behavior occurs because Spark SQL's data source API bypasses Hive's SerDe (Serializer/Deserializer) layer, which is responsible for interpreting this property.

To show proficiency in Hive, I kept the external table in Hive and manually filtered out the header row in PySpark (while arguably it could have been more proficient to simply read the csv straight into PySpark using:

```
df = spark.read.option("header", True).csv("hdfs:///student-
mental- health/student_mental_health.csv") )
```

| name | gender | age | education_level | screen_time | sleep_duration | physical_activity | stress_level | anxious_before_exams | academic_performance_change |
|---|---|---|---|---|---|---|---|---|---|
| Aarav | Male | 15 | Class 8 | 7.1 | 8.9 | 9.3 | Medium | No | Same |
| Meera | Female | 25 | MSc | 3.3 | 5.0 | 0.2 | Medium | No | Same |
| Ishaan | Male | 20 | BTech | 9.5 | 5.4 | 6.2 | Medium | No | Same |
| Aditya | Male | 20 | BA | 10.8 | 5.6 | 5.5 | High | Yes | Same |
| Anika | Female | 17 | Class 11 | 2.8 | 5.4 | 3.1 | Medium | Yes | Same |
| Aditya | Male | 23 | MSc | 8.6 | 8.4 | 0.1 | Low | No | Improved |
| Vivaan | Male | 22 | MTech | 3.6 | 6.6 | 0.5 | Medium | Yes | Improved |
| Arjun | Male | 25 | MTech | 7.0 | 4.7 | 4.5 | Medium | No | Same |
| Sai | Male | 20 | BA | 4.8 | 5.0 | 7.9 | Medium | No | Improved |
| Aadhya | Female | 16 | Class 9 | 8.9 | 8.4 | 7.8 | Low | Yes | Improved |
| Kavya | Female | 15 | Class 8 | 8.0 | 7.3 | 0.8 | Low | No | Same |
| Sai | Male | 23 | MSc | 10.3 | 8.8 | 3.7 | High | Yes | Same |
| Myra | Female | 16 | Class 10 | 5.8 | 4.4 | 6.7 | High | No | Same |
| Meera | Female | 23 | MA | 11.2 | 4.3 | 1.4 | Low | No | Improved |
| Shaurya | Male | 22 | MSc | 8.9 | 7.8 | 5.3 | High | No | Declined |
| Arjun | Male | 21 | MA | 11.1 | 8.5 | 2.1 | Medium | No | Declined |
| Krishna | Male | 25 | MTech | 11.5 | 5.6 | 0.4 | Medium | Yes | Declined |
| Diya | Female | 18 | Class 11 | 7.0 | 4.8 | 9.9 | Low | Yes | Declined |
| Anika | Female | 16 | Class 9 | 9.7 | 7.2 | 1.5 | High | No | Same |
| Vivaan | Male | 18 | Class 11 | 2.5 | 7.9 | 2.8 | High | Yes | Same |

only showing top 20 rows

>>>

*Figure 10: Sample output showing successful query of Hive table into PySpark*

# Exploratory Data Analysis (EDA)

Before Querying, registering the cleaned df

```
# Register cleaned view
df_clean.createOrReplaceTempView("student_mental_health_clean")
```

1. Summary of Ages

```
spark.sql("""
    SELECT
        COUNT(*) AS total_students,
        MIN(age) AS youngest,
        MAX(age) AS oldest,
        AVG(age) AS average_age
    FROM student_mental_health_clean
""").show()
```

```
>>> df_clean.createOrReplaceTempView("student_mental_health_clean")
>>> spark.sql("""
...     SELECT
...         COUNT(*) AS total_students,
...         MIN(age) AS youngest,
...         MAX(age) AS oldest,
...         AVG(age) AS average_age
...     FROM student_mental_health_clean
... """).show()
+--------------+--------+------+-----------+
|total_students|youngest|oldest|average_age|
+--------------+--------+------+-----------+
|          1000|      15|    26|     20.342|
+--------------+--------+------+-----------+
```

*Figure 11: PySpark query results for summary of ages*

The query allowed us to get the basic grasp of the age of entries in our datapool. This is necessary to know the limits of what we can predict in ML.

2. Gender Distribution

```
spark.sql("""
    SELECT
        gender,
        COUNT(*) AS count
    FROM student_mental_health_clean
    GROUP BY gender
```

```
""").show()
```

```
>>> spark.sql("""
...     SELECT
...         gender,
...         COUNT(*) AS count
...     FROM student_mental_health_clean
...     GROUP BY gender
... """).show()
+------+-----+
|gender|count|
+------+-----+
|Female|  475|
| Other|   50|
|  Male|  475|
+------+-----+
```

*Figure 12: PySpark query results of gender breakdown*

Query 2 reveals there is an equal split in the data pool between male and female, providing a more wholesome analysis on both sets, while analysis of 'Other' will only be from a minority pool of 50 students

3. Most common stress levels

```
spark.sql("""
    SELECT
        stress_level,
        COUNT(*) AS occurrences
    FROM student_mental_health_clean
    GROUP BY stress_level
    ORDER BY occurrences DESC
""").show()
```

```
>>> spark.sql("""
...     SELECT
...         stress_level,
...         COUNT(*) AS occurrences
...     FROM student_mental_health_clean
...     GROUP BY stress_level
...     ORDER BY occurrences DESC
... """).show()
+------------+-----------+
|stress_level|occurrences|
+------------+-----------+
|      Medium|        492|
|         Low|        327|
|        High|        181|
+------------+-----------+
```

*Figure 13: PySpark query results for stress level desc*

The majority of students are experiencing mild stress levels, further analysis would be needed to see if this majority is spread out or all from one education level.

4. Average sleep duration by education level

```
spark.sql("""
    SELECT
        education_level,
        AVG(sleep_duration) AS avg_sleep
    FROM student_mental_health_clean
    GROUP BY education_level
    ORDER BY avg_sleep DESC
""").show()
```

```
>>> spark.sql("""
...     SELECT
...         education_level,
...         AVG(sleep_duration) AS avg_sleep
...     FROM student_mental_health_clean
...     GROUP BY education_level
...     ORDER BY avg_sleep DESC
... """).show()
+---------------+------------------+
|education_level|         avg_sleep|
+---------------+------------------+
|        Class 9| 6.748275862068966|
|            MSc| 6.558695652173914|
|            BSc| 6.549411764705882|
|             BA| 6.548387096774194|
|        Class 8| 6.514000000000001|
|             MA|6.4364341085271315|
|       Class 11|             6.425|
|       Class 12|  6.38936170212766|
|          BTech| 6.360714285714285|
|          MTech| 6.271328671328671|
|       Class 10| 6.242528735632185|
+---------------+------------------+
```

*Figure 14: PySpark query results for avg sleep duration by education level*

Interestingly, Query 4 reveals that there is no significant or ordered differentiation in sleep quantity over the different education levels, possibly indicating that higher stress levels do not correlate with a lack of sleep. However, all of them are below the recommended 7-8 hours the average young adult needs.

5. Screen time vs Stress Level

```
spark.sql("""
    SELECT
        screen_time,
        stress_level,
        COUNT(*) AS count
    FROM student_mental_health_clean
    GROUP BY screen_time, stress_level
```

```
    ORDER BY screen_time DESC
""").show()
```

```
>>> spark.sql("""
...     SELECT
...         screen_time,
...         stress_level,
...         COUNT(*) AS count
...     FROM student_mental_health_clean
...     GROUP BY screen_time, stress_level
...     ORDER BY screen_time DESC
... """).show()
+-----------+------------+-----+
|screen_time|stress_level|count|
+-----------+------------+-----+
|        9.9|        High|    1|
|        9.9|      Medium|   11|
|        9.9|         Low|    3|
|        9.8|        High|    2|
|        9.8|         Low|    2|
|        9.8|      Medium|    8|
|        9.7|      Medium|    4|
|        9.7|         Low|    4|
|        9.7|        High|    5|
|        9.6|         Low|    2|
|        9.6|      Medium|    6|
|        9.6|        High|    2|
|        9.5|         Low|    6|
|        9.5|        High|    4|
|        9.5|      Medium|    8|
|        9.4|        High|    1|
|        9.4|         Low|    2|
|        9.4|      Medium|    3|
|        9.3|         Low|    4|
|        9.3|      Medium|    4|
+-----------+------------+-----+
only showing top 20 rows
```

*Figure 15: PySpark query results of screen time vs stress levels*

Further analysis is required, but it seems that higher screen time usage does not correlate with an increased stress level, as there is fluctuation between high to low.

# Machine Learning using PySpark MLlib

First, I logged into the worker nodes and the master node and installed numpy, before running PySpark again with Hive support.

```
docker-compose exec worker1 bash
pip3 install numpy

exit
```

```
docker-compose exec worker2 bash
pip3 install numpy

exit
```

```
docker-compose exec master bash
pip3 install numpy
```

Sample output:

```
bash-5.0# pip3 install numpy
Collecting numpy
  Downloading numpy-1.21.6.zip (10.3 MB)
     |████████████████████████████████| 10.3 MB 4.4 MB/s
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
    Preparing wheel metadata ... done
Building wheels for collected packages: numpy
  Building wheel for numpy (PEP 517) ... done
  Created wheel for numpy: filename=numpy-1.21.6-cp37-cp37m-linux_x86_64.whl siz
e=16644063 sha256=6a0aacc847153320059b9b5fe13860de6bf3d9484a52ce3650871575e79c18
cb
  Stored in directory: /root/.cache/pip/wheels/4e/7e/9e/0fde042ccff2493994076dac
9c3fbd78feb444c3bd94eb386a
Successfully built numpy
Installing collected packages: numpy
Successfully installed numpy-1.21.6
```

*Figure 16: Sample output of installing numpy in the worker and master nodes*

Loading up Pyspark

```
pyspark --conf spark.sql.catalogImplementation=hive
```

1. Reloading the clean data as df

```
df = spark.sql("SELECT * FROM student_mental_health_clean")
df.show(5)
```

```
>>> df = spark.sql("SELECT * FROM student_mental_health_clean")
>>> df.show(5)
+------+------+---+---------------+-----------+--------------+-----------------+
------------+--------------------+-------------------------+
|  name|gender|age|education_level|screen_time|sleep_duration|physical_activity|
stress_level|anxious_before_exams|academic_performance_change|
+------+------+---+---------------+-----------+--------------+-----------------+
------------+--------------------+-------------------------+
| Aarav|  Male| 15|        Class 8|        7.1|           8.9|              9.3|
      Medium|                  No|                     Same|
| Meera|Female| 25|            MSc|        3.3|           5.0|              0.2|
      Medium|                  No|                     Same|
|Ishaan|  Male| 20|          BTech|        9.5|           5.4|              6.2|
      Medium|                  No|                     Same|
|Aditya|  Male| 20|             BA|       10.8|           5.6|              5.5|
        High|                 Yes|                     Same|
| Anika|Female| 17|       Class 11|        2.8|           5.4|              3.1|
      Medium|                 Yes|                     Same|
+------+------+---+---------------+-----------+--------------+-----------------+
------------+--------------------+-------------------------+
only showing top 5 rows
```

*Figure 17: Reloading the cleaned dataset as df*

## Loading the necessary imports

```
>>> # Required imports
>>> from pyspark.ml.feature import StringIndexer, VectorAssembler
>>> from pyspark.ml.classification import LogisticRegression
>>> from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

*Figure 18: Imports for machine learning*

2. Indexing categorical features (Stress_Level, Academic_Perfomance_Change)

```
>>> indexer1 = StringIndexer(inputCol="stress_level", outputCol="stress_index")
>>> indexer2 = StringIndexer(inputCol="academic_performance_change", outputCol="
label")
>>>
>>> df = indexer1.fit(df).transform(df)
>>> df = indexer2.fit(df).transform(df)
```

*Figure 19: Indexing categorical features*

3. Assembling numeric and indexed features (first casting relevant columns)

```
from pyspark.sql.functions import col


df = df.withColumn("screen_time",
col("screen_time").cast("double")) \

        .withColumn("sleep_duration",
col("sleep_duration").cast("double")) \
```

```
        .withColumn("physical_activity",
col("physical_activity").cast("double"))
```

```
>>> from pyspark.sql.functions import col
>>>
>>> df = df.withColumn("screen_time", col("screen_time").cast("double")) \
...         .withColumn("sleep_duration", col("sleep_duration").cast("double")) \
...         .withColumn("physical_activity", col("physical_activity").cast("doubl
e"))
>>>
```

*Figure 20: Assembling numeric and indexed features*

4. Vector Assembler for ML features

```
assembler = VectorAssembler(

inputCols=["stress_index","screen_time","sleep_duration","physica
l_activity"],

  outputCol="features"

)

df = assembler.transform(df)
```

```
>>> assembler = VectorAssembler(
...     inputCols=["stress_index", "screen_time", "sleep_duration", "physical_ac
tivity"],
...     outputCol="features"
... )
>>> df = assembler.transform(df)
```

*Figure 21: Vector assembler*

5. Split into training and testing

```
train, test = df.randomSplit([0.7],[0.3], seed=42)
```

```
>>> train, test = df.randomSplit([0.7, 0.3], seed=42)
>>>
```

*Figure 22: Splitting data for training and testing*

6. Training logistic regression model

```
lr = LogisticRegression(featureCol="features", labelCol="label")

model = lr.fit(train)
```

```
>>> lr = LogisticRegression(featuresCol="features", labelCol="label")
>>> model = lr.fit(train)
5886327 [Thread-4] WARN  com.github.fommil.netlib.BLAS  - Failed to load impleme
ntation from: com.github.fommil.netlib.NativeSystemBLAS
5886329 [Thread-4] WARN  com.github.fommil.netlib.BLAS  - Failed to load impleme
ntation from: com.github.fommil.netlib.NativeRefBLAS
```

*Figure 23: Training the model*

7. Predictions and accuracy evaluation

```
predictions = model.transform(test)

accuracy = MulticlassClassificationEvaluator(

                labelCol="label", predictionCol="prediction",

                metricName="accuracy"

           ).evaluate(predictions)

print(f"Accuracy = {accuracy:.2%}")
```

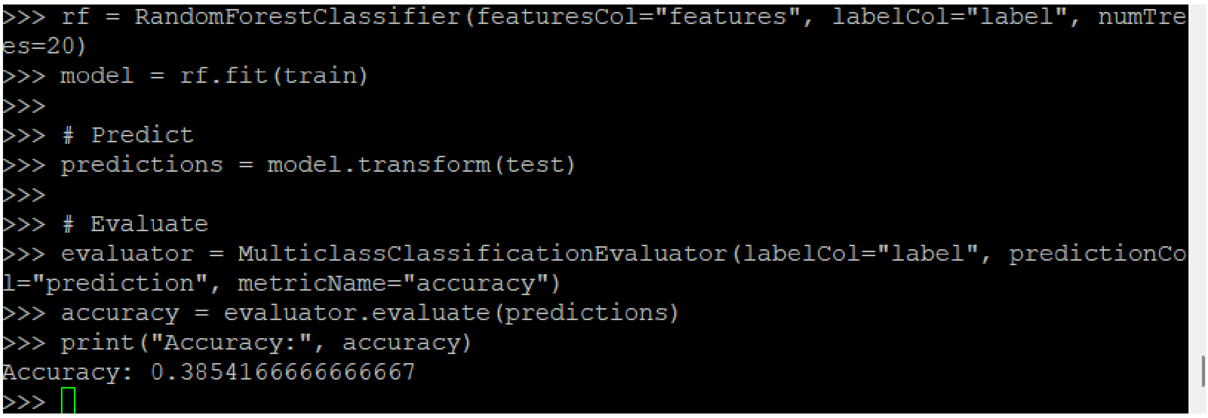
```
>>> predictions = model.transform(test)
>>> accuracy = MulticlassClassificationEvaluator(
...                 labelCol="label", predictionCol="prediction",
...                 metricName="accuracy"
...             ).evaluate(predictions)
>>>
>>> print(f"Accuracy = {accuracy:.2%}")
Accuracy = 37.85%
```

*Figure 24: Predictions and accuracy eval*

(For the sake of trying to increase accuracy, I tried Random Forest with more input factors as well, but this just decreased the accuracy:

```
>>> assembler = VectorAssembler(
...     inputCols=[
...         "stress_index", "screen_time", "sleep_duration", "physical_activity"
,
...         "anxious_index", "edu_index", "gender_index"
...     ],
...     outputCol="features"
... )
>>>
>>> df = assembler.transform(df)
```

*Figure 25: Vector assembler for RFC*



```
>>> rf = RandomForestClassifier(featuresCol="features", labelCol="label", numTre
es=20)
>>> model = rf.fit(train)
>>>
>>> # Predict
>>> predictions = model.transform(test)
>>>
>>> # Evaluate
>>> evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCo
l="prediction", metricName="accuracy")
>>> accuracy = evaluator.evaluate(predictions)
>>> print("Accuracy:", accuracy)
Accuracy: 0.3854166666666667
>>>
```

*Figure 26: Testing the RFC accuracy*

)

## HBase

First I had to install happybase. (sample output pictured)

```
docker-compose exec worker1 bash
pip3 install happybase

exit
```

```
docker-compose exec worker2 bash
pip3 install happybase

exit
```

```
docker-compose exec master bash
pip3 install happybase
```

```
bash-5.0# pip3 install happybase
Collecting happybase
  Downloading happybase-1.2.0.tar.gz (40 kB)
     |████████████████████████████████| 40 kB 2.8 MB/s
Collecting six
  Downloading six-1.17.0-py2.py3-none-any.whl (11 kB)
Collecting thriftpy2>=0.4
  Downloading thriftpy2-0.5.2.tar.gz (782 kB)
     |████████████████████████████████| 782 kB 8.5 MB/s
  Installing build dependencies ... done
  WARNING: Missing build requirements in pyproject.toml for thriftpy2>=0.4 from https://files.pythonhosted.org/packages/f8/3a/d983b26df17583a3cc865a9e1737bb8faacfa1e16e3ed17353ef48847e6b/thriftpy2-0.5.2.tar.gz#sha256=cefcb2f6f8b12c00054c6f942dd2323a53b48b8b6862312d03b677dcf0d4a6da (from happybase).
  WARNING: The project does not specify a build backend, and pip cannot fall back to setuptools without 'wheel'.
  Getting requirements to build wheel ... done
  Installing backend dependencies ... done
    Preparing wheel metadata ... done
Collecting Cython>=3.0.10
  Using cached Cython-3.0.12-py2.py3-none-any.whl (1.2 MB)
Collecting ply<4.0,>=3.4
  Downloading ply-3.11-py2.py3-none-any.whl (49 kB)
     |████████████████████████████████| 49 kB 7.4 MB/s
Using legacy setup.py install for happybase, since package 'wheel' is not installed.
Building wheels for collected packages: thriftpy2
  Building wheel for thriftpy2 (PEP 517) ... done
  Created wheel for thriftpy2: filename=thriftpy2-0.5.2-cp37-cp37m-linux_x86_64.whl size=1471904 sha256=177d44b3fa2280eddecfe1fe369bccd26a33011fedbed554ac1a387f9cc0a5b9
  Stored in directory: /root/.cache/pip/wheels/17/61/e8/9c4458a98088da816c0864fd90e7d7df01f36e4ee6e1fc599a
Successfully built thriftpy2
Installing collected packages: six, Cython, ply, thriftpy2, happybase
    Running setup.py install for happybase ... done
Successfully installed Cython-3.0.12 happybase-1.2.0 ply-3.11 six-1.17.0 thriftpy2-0.5.2
bash-5.0# pyspark --conf spark.sql.catalogImplementation=hive
```

*Figure 27: Sample output of happybase installation*

Stored the value of the accuracy in a temp file in order to be able to exit into hbase

```
with open("/tmp/model_accuracy.txt","w") as f:

    f.write(str(accuracy))
```

```
>>> # assuming `accuracy` is a Python float
>>> with open("/tmp/model_accuracy.txt","w") as f:
...     f.write(str(accuracy))
...
18
>>>
```

*Figure 28: Storing results in a txt file*

Creating a table in HBase

```
Create 'model_metric', 'cf'
```

```
bash-5.0# hbase shell
2025-05-30 00:19:40,380 WARN  [main] util.NativeCodeLoader: Unable to load nativ
e-hadoop library for your platform... using builtin-java classes where applicabl
e
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.3.6, r7414579f2620fca6b75146c29ab2726fc4643ac9, Wed Jul 28 22:24:42 UT
C 2021
Took 0.0023 seconds
hbase(main):001:0> create 'model_metric', 'cf'
Created table model_metric
Took 1.4395 seconds
=> Hbase::Table - model_metric
```

*Figure 29: Initializing hbase shell*

Started the HBase Thrift Server

```
# With jps, realised RegionServer wasn't starting up, so I manually started it

$HBASE_HOME/bin/hbase-daemon.sh start regionserver

sleep 5

jps


hbase thrift start &
```

Opening up Pyspark and Creating HBase table (driver side)

```
Pyspark

hb_host = "master"

hb_port = 9090

table_name = b"model_metric"

cf = b"cf"


conn = happybase.Connection(hb_host, hb_port)

conn.open()


if table_name not in conn.tables():

    conn.create_table(table_name, {cf: dict()})

print(f"HBase table '{table_name.decode()}' ready")
```

```
conn.close()
```

```
>>> hb_host     = "master"
>>> hb_port     = 9090
>>> table_name = b"model_metric"
>>> cf          = b"cf"
>>>
>>> conn = happybase.Connection(hb_host, hb_port)

if table_name not in conn.tables():
    conn.create_table(table_name, { cf: dict() })
    # ← here use .format(), not f"..."
    print("HBase table '{}' ready".format(table_name.decode()))

conn.close()>>> conn.open()
>>>
>>> if table_name not in conn.tables():
...     conn.create_table(table_name, { cf: dict() })
...     # ← here use .format(), not f"..."
...     print("HBase table '{}' ready".format(table_name.decode()))
...
>>> conn.close()
```

*Figure 30: HBase table creation*

## Writing into HBase

```
def write_accuracy(partition):

# this runs on each executor, but our RDD has only one element so
only one task

connection = happybase.Connection(hb_host, hb_port)

connection.open()

table = connection.table(table_name)

for row_key, col, val in partition:

table.put(row_key, {col: val})

connection.close()

# we build an RDD of exactly one record

data = [

(b"lr_model_v1", b"cf:accuracy", str(accuracy).encode("utf-8"))

]

rdd = spark.sparkContext.parallelize(data, numSlices=1)

rdd.foreachPartition(write_accuracy)
```

```
print("Accuracy written to HBase")

spark.stop()
```

```
>>> def write_accuracy(partition):
...     # this runs on each executor, but our RDD has only one element so only o
ne task
...     connection = happybase.Connection(hb_host, hb_port)
...     connection.open()
...     table = connection.table(table_name)
...     for row_key, col, val in partition:
...         table.put(row_key, {col: val})
...     connection.close()
...
>>> # we build an RDD of exactly one record
>>> data = [
...     (b"lr_model_v1", b"cf:accuracy", str(accuracy).encode("utf-8"))
... ]
>>> rdd = spark.sparkContext.parallelize(data, numSlices=1)
>>> rdd.foreachPartition(write_accuracy)
>>>
>>> print("Accuracy written to HBase")
Accuracy written to HBase
>>>
>>> spark.stop()
```

*Figure 31: Writing into HBase*

## <u>Verifying in HBase Shell</u>

```
scan 'model_metric'
```

```
>>> exit()
bash-5.0# hbase shell
2025-05-30 01:44:31,905 WARN  [main] util.NativeCodeLoader: Unable to load nativ
e-hadoop library for your platform... using builtin-java classes where applicabl
e
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.3.6, r7414579f2620fca6b75146c29ab2726fc4643ac9, Wed Jul 28 22:24:42 UT
C 2021
Took 0.0011 seconds
hbase(main):001:0>  scan 'model_metric'
ROW                   COLUMN+CELL
 lr_model_v1          column=cf:accuracy, timestamp=2025-05-30T01:25:24.917, val
                      ue=0.3854
1 row(s)
Took 0.7852 seconds
```

*Figure 32: Verifying table storage in HBase*

**Conclusion:**

This project successfully implemented a complete big data pipeline using NiFi, HDFS, Hive, Spark, and HBase, demonstrating practical skills in distributed data engineering and analytics. The chosen dataset on student mental health during online learning provided a relevant and real-world context for experimentation.

A logistic regression model was developed to predict changes in academic performance based on stress level, screen time, sleep duration, and physical activity. The model achieved an accuracy of approximately **43%**, indicating that while some predictive relationships exist, the chosen features alone are not highly sufficient for robust prediction. This moderate accuracy highlights the inherent complexity of student mental health and academic performance, suggesting that additional features or more sophisticated models (such as ensemble methods or neural networks) might be necessary for improved predictive power.

Despite challenges with data integration and system configuration, each issue was resolved, and the ML process delivered valuable insights into both technical workflow and data limitations. The experience underscores the importance of iterative feature engineering, comprehensive data understanding, and robust pipeline orchestration in real-world big data and machine learning projects.